Contents

1 Introduction

Minesweeper is a widely recognised computer game, best known for being included in the standard install of Windows from versions 3.1 to 7, and was first available as part of *Windows Entertainment Pack 1* as early as 1990. The main mathematical interest it has received is the work of *Kaye*[1] in which he proves using a constructive method that it is NP-complete to determine whether a given minesweeper grid has a possible arrangement of mines within the rules of the game.

Minesweeper is a single-player game, with the aim being to complete the board as quickly as possible by clicking all cells which don't contain a mine, while avoiding clicking any cells which contain a mine which would lead to a game-over. The board is a rectangular grid of square cells, initially all blank. When a safe cell is clicked a number is revealed, with this number corresponding to the exact number of mines contained in the surrounding eight cells (unless none of the surrounding cells contain a mine, so the number would be 0, in which case no number is displayed but the surrounding cells are automatically selected, creating an 'opening'). Using the displayed numbers it is possible to deduce where the mines must be and which cells must be safe, meaning that after an opening is found it is often (but not always) possible to complete a board without any risk of a game-over. The higher the density of mines, the more likely it is that there will be insufficient information available through the numbers displayed to deduce any safe cells, and the player must rely on luck to find a safe cell providing required information to allow for continued logical deduction.

In more recent editions of the game it is guaranteed that the first click will reveal an opening, removing the initial dependence on luck. In the original game you are required to

find an opening using luck, however there is some strategy that can be used to maximise the chance of finding one. For example clicking a cell in a corner of the board only requires the surrounding three cells to be safe for an opening to be found, however a central cell would require all eight surrounding cells to be safe, which has a much lower probability. On the other hand, an opening in the centre of the board will be bigger on average, and therefore more useful to get started on a game. When looking for an opening there is a fairly high chance of revealing a safe cell containing a number, which can indicate which cells are more likely to be safe. For example revealing a number 1 on the first click means each surrounding cell contains a mine with probability 1/8, which tends to be lower than the probability of the other isolated cells being safe (usually about 20%). With multiple numbers exposed in adjacent cells, calculating the probability of the surrounding cells being safe becomes more complex, but could be used to increase the chances of finding an opening without hitting a mine.

In this work the aim is to find a set of equations which can be used to calculate the probabilities of each cell containing a mine for any partially completed minesweeper board. This could be applied in both of the above scenarios – knowing which cells are safer when there are no guaranteed safe cells left would lead to less frustrating losses towards the end of a game, and having a strategy for finding an opening at the start of a game would also clearly be beneficial.

2 Definitions and standard values

We will be considering various minesweeper boards and configurations within them. Before proceeding we first define the following parameters and variables:

- N, total number of cells;
- K, total number of mines;
- ρ, density of mines;
- $\rho_0 := K/N$, initial density of mines;
- $x_{\max}$, maximum allowed number of mines in a single cell.

Note that in the standard minesweeper game $x_{\max}$ was always taken to be 1 (there was never more than 1 mine in each cell).

The parameter values at the start of a game for the standard difficulties are given in Table 1.

Difficulty	Dimensions	N	K	ρ_0
Beginner (b)	8×8	64	10	0.156
Intermediate (i)	16×16	256	40	0.156
Expert (e)	16×30	480	99	0.206
Master (m)	30×30	900	200	0.222

Table 1: Characteristics of standard games.

3 Probabilities by combinatorics

3.1 One mine per cell

We initially consider the classic game, where each cell may contain either zero or one mine ($x_{\max} = 1$). We calculate the probabilities of different configurations occurring by considering the number of ways of placing the mines to create such a configuration. To demonstrate how we perform this count, consider a new game – a grid with N cells and K mines. Note that there is no need to distinguish between the K mines, so the number of ways that the mines could be arranged, which we call q, is given by the binomial coefficient

$$q = \binom{N}{K} = \frac{N!}{(N-K)!K!}. \tag{3.1}$$

Now suppose a cell somewhere in the middle of the board is clicked, revealing a number α, where of course $0 \leq \alpha \leq 8$. We then know the number of mines in the surrounding eight cells is exactly α, and there are $(K - \alpha)$ mines in the remaining $(N - 9)$ cells. The number of ways to arrange α mines in eight cells is $\binom{8}{\alpha}$ and the number of arrangements for the remaining mines is $\binom{N-9}{K-\alpha}$, giving a total number of arrangements

$$q = \binom{N-9}{K-\alpha}\binom{8}{\alpha}. \tag{3.2}$$

Notice that in this case the eight cells surrounding the revealed number all have the same probability of containing a mine because they share exactly the same neighbouring information provided by the single revealed number. Therefore we say these eight cells are in the same 'equivalence group', and that in this case the board has just one equivalence group (we don't count the cells which have no neighbouring numbers displayed, although they are also technically an equivalence group). The size of the equivalence group is $s = 8$, and it contains $m = \alpha$ mines. If we can find the probability distribution for the number of mines in an equivalence group, we can then determine the probability of each cell containing a mine.

We may use this to calculate the probability distribution for the value of the number revealed, α. We calculate the probabilities by comparing the number of configurations of mines after α is exposed to the number of configurations before. The probability of α being shown is then

$$p_\alpha = \binom{N-9}{K-\alpha}\binom{8}{\alpha} \Bigg/ \binom{N}{K} = \binom{8}{\alpha}\frac{(N-K)_{9-\alpha}(K)_\alpha}{(N)_9}, \tag{3.3}$$

where $(x)_n$ denotes the falling factorial $x!/(x-n)!$. For an expert grid with $N = 480$ and $K = 99$ the probabilities for the number being α are

$$p_0 = 12.3\%, \quad p_1 = 26.0\%, \quad p_2 = 23.9\%, \quad p_3 = 12.4\%, \quad p_4 = 4.9\%,$$

which may be compared to the probability of hitting a mine, $p_m = 20.6\%$.[1]

We can generalise the case of there being one equivalence group to having r equivalence groups, where group i has size s_i and contains m_i mines (where we assume for now that

[1] As a comparison, on a beginner grid the probabilities are $p_0 = 19.3\%$, $p_1 = 33.6\%$, $p_2 = 22.5\%$, $p_3 = 7.5\%$, $p_4 = 1.3\%$ and $p_m = 15.6\%$.

the number of mines in each equivalence group is known). We define the total number of cells and mines contained in all cells with an adjacent exposed number to be $S = \sum_{i=1}^{r} s_i$ and $M = \sum_{i=1}^{r} m_i$ respectively, and N' to be the total number of cells which haven't been clicked. The number of arrangements is then

$$q = \binom{N' - S}{K - M} \prod_{i=1}^{r} \binom{s_i}{m_i}. \tag{3.4}$$

If we are given a board layout, we will not necessarily know the exact number of mines in each equivalence group. We may form a set of possible configurations, in the form of r-tuples containing the number of mines in each of the r groups. For example in the simple '1-2' situation shown in Figure 1, the set of possibilities would be $\{(1, 0, 2), (0, 1, 1)\}$. We

(a) Configuration of mines $(1, 0, 2)$. (b) Configuration of mines $(0, 1, 1)$.

Figure 1: Groups of equivalent cells represented by colours (numbered from left to right), mines represented by asterisks.

denote the number of ways to arrange the mines in configuration j to be q_j, and the total number of configurations to be $Q = \sum_j q_j$. Then the probability of configuration j occurring is simply the ratio of the number of arrangements of mines in this configuration to the total number of possible arrangements,

$$P(C_j) = \frac{q_j}{Q} = \frac{1}{Q} \binom{N' - S}{K - M_j} \prod_{i=1}^{r} \binom{s_i}{m_{ij}}, \tag{3.5}$$

where m_{ij} is the number of mines in group i, configuration j, and $M_j = \sum_{i=1}^{r} m_{ij}$ is the total number of mines in the main groups in configuration j. From this we can find the probability distribution of the number of mines in each equivalence group, and consequently the probability of each cell containing a mine. We see that there are factors which cancel in the probability calculation (at the very least the factor Q), and we can simplify by defining a relative probability of C_j. This can be written in many ways, one which involves fewer terms in the factorial is

$$\xi(C_j) = (K)_{M_j} (N' - S - K + \hat{M})_{\hat{M} - M_j} \prod_{i=1}^{r} \binom{s_i}{m_{ij}}, \tag{3.6}$$

where $\hat{M} = \max_j M_j$. The probability is then given by

$$P(C_i) = \frac{\xi(C_i)}{\sum_{j=1}^{r} \xi(C_j)}. \tag{3.7}$$

3.2 Multiple mines per cell

We now allow the maximum number of mines in a cell to be greater than one, $x_{\max} > 1$. It is now slightly more complicated to count the number of arrangements of mines, since

different arrangements have different probabilities of occurring – there are two ways for two mines to be placed in different cells but only one way for them to both be placed in one cell. For this reason we now need to treat the mines as distinct, that is there are four ways to place two mines in two cells because there are two ways to place one mine in each.

We first check this gives us the same result for $x_{\max} = 1$. Consider again the case that one number in the centre is revealed to be α, then the number of ways to place the α distinct mines in the group of eight surrounding cells is $(8)_\alpha$ (rather than $\binom{8}{\alpha}$). We must also consider the number of ways to choose the α mines out of the K total mines, which there are $\binom{K}{\alpha}$ ways to do. There are then $(K-\alpha)$ mines left to place in the remaining $(N-9)$ cells, which has $(N-9)_{K-\alpha}$ arrangements. This gives the total number of arrangements

$$q = \frac{8!}{(8-\alpha)!} \cdot \frac{K!}{\alpha!(K-\alpha)!} \cdot \frac{(N-9)!}{(K-9-K+\alpha)!} = K! \binom{8}{\alpha} \binom{N-9}{K-\alpha}, \tag{3.8}$$

which differs to (3.2) only by the additional $K!$ term, which is due to the distinction of the K mines, and this term will cancel out when calculating the probabilities. For r groups this generalises to

$$\begin{aligned} q &= (s_1)_{m_1} \binom{K}{m_1} (s_2)_{m_2} \binom{K-m_1}{m_2} \dots (s_r)_{m_r} \binom{K-\sum_{i=1}^{r-1} m_i}{m_r} (N'-S)_{K-M} \\ &= (N'-S)_{K-M} \binom{K}{m_1, m_2, \dots, m_r, K-M} \prod_{i=1}^{r} (s_i)_{m_i} \\ &= K! \frac{(N'-S)_{K-M}}{(K-M)!} \prod_{i=1}^{r} \frac{(s_i)_{m_i}}{m_i!}. \end{aligned}$$

We could simplify with the binomial notation to arrive at an analogous expression to (3.4), however the distinction made here will allow us to extend to multiple mines per cell, since the falling factorial $(s_i)_{m_i}$ is the number of ways to arrange m_i distinct mines in s_i cells.

Using the above formulation, we can write the number of arrangements as

$$q = K! \frac{\tilde{w}}{(K-M)!} \prod_{i=1}^{r} \frac{w_i}{m_i!}, \tag{3.9}$$

where w_i is the number of ways to arrange the mines in group i and $\tilde{w}$ is the number of ways to arrange the mines in the outer group. We are then able to redefine ξ for multiple mines per cell as

$$\xi(C_j) = \frac{\tilde{w}_j}{(K-M_j)!} \prod_{i=1}^{r} \frac{w_{ij}}{m_{ij}!}, \tag{3.10}$$

where (3.7) still holds.

The task we are left with is calculating the values of w_{ij} and $\tilde{w}_j$. We consider a general group of size s, containing m mines. If $m \leq x_{\max}$ each cell can contain any number of the m mines, so each mine has s places it can be, giving s^m arrangements. This provides an upper bound for any value of $x_{\max}$. There is no simple formula for the case $m > x_{\max}$, however we are able to formulate a process to count the number of arrangements. To illustrate the method, consider the example of $s = 8$, $m = 6$ and $x_{\max} = 3$. The arrangements are then:

- $\binom{8}{2} \times \binom{6}{3}$ ways to arrange $(3,3,0,0,0,0,0,0)$,
- $\binom{8}{1,1,1,5} \times \binom{6}{3,2,1}$ ways to arrange $(3,2,1,0,0,0,0,0)$,
- $\binom{8}{1,3,4} \times \binom{6}{3,1,1,1}$ ways to arrange $(3,1,1,1,0,0,0,0)$,
- $\binom{8}{3} \times \binom{6}{2,2,2}$ ways to arrange $(2,2,2,0,0,0,0,0)$,
- $\binom{8}{2,2,4} \times \binom{6}{2,2,1,1}$ ways to arrange $(2,2,1,1,0,0,0,0)$,
- $\binom{8}{1,4,3} \times \binom{6}{2,1,1,1,1}$ ways to arrange $(2,1,1,1,1,0,0,0)$,
- the usual $\binom{8}{6} \times 6!$ ways to arrange $(1,1,1,1,1,1,0,0)$.

The first arrangement is found by placing mines in the group furthest to the left that has space. The other arrangements are found by shifting mines into groups further right without creating any copies. Using the first arrangement as an example, $\binom{8}{2}$ is the number of ways to arrange the two number '3's in the eight positions, and $\binom{6}{3}$ is the number of ways to place the unique six mines in groups of three and three. This logic extends naturally for the other arrangements. These can be summed to give the total number of arrangements of the group, which in this case gives $w = 255,920$.

The largest possible inner group will be of size 8, so it is a simple matter to calculate all the required arrangements. In fact in this case there will be a fixed number of mines in the group anyway and for variable number of mines the largest possible groups size is 7, where only a corner cell is shared between two numbers. It is not as simple for the outer group, since the group size could easily be as large as 500. A standard computer is capable of computing the number of arrangements in a few seconds for large groups, however in practice we would like the performance to be much quicker.

We would now like to form a mathematical equation for the number of arrangements, w. For $x_{\max} = 1$ we already know the equation is $w = (s)_m$. To begin with we extend this for $x_{\max} = 2$, and we will then use this to seek a formula for larger $x_{\max}$. We note that arranging m mines with d double-mine cells can be done in $m!/2^d$ ways, and the number of ways to arrange the numbers in the s positions is $s!/(d!(m-2d)!(s-m+d)!)$. Finally we need to sum over all valid choices for d, the number of double mines, which will be limited above by the number of mines, m, and below by the number of cells, s. The upper limit is $\lfloor m/2 \rfloor$, and if $s \geq m$, $d = 0$ is valid (there is space for the mines all to be in their own cells), otherwise the lower limit is $m - s$. We can then write down the total number of ways to arrange mines in a general set of cells with $x_{\max} = 2$:

$$w = \sum_{d=\max(0,m-s)}^{\lfloor m/2 \rfloor} \frac{s!m!}{2^d d!(m-2d)!(s-m+d)!}. \tag{3.11}$$

Note the $d = 0$ term gives $(s)_m$ as expected. To extend this method to $x_{\max} = 3$ we would introduce another sum for the number of triple-mine cells. Let there be t triple-mine cells, where $\max(0, m-2s) \leq t \leq \lfloor m/3 \rfloor$. For each triple-mine cell there is one less cell and three less mines to consider for the sum over double-mine cells, so we replace m with $m-3t$ and s with $s-t$ in the limits. We then have

$$w = \sum_{t=\max(0,m-2s)}^{\lfloor m/3 \rfloor} \sum_{d=\max(0,m-2t-s)}^{\lfloor (m-3t)/2 \rfloor} \frac{s!m!}{2^d (3!)^t d!t!(m-2d-3t)!(s-m+d+2t)!}. \tag{3.12}$$

This can be extended for higher $x_{\max}$, however it becomes more messy. As a further illustration of how this is done, the equation for $x_{\max} = 4$ is given below:

$$w = \sum_{f=\max(0,m-3s)}^{\lfloor m/4 \rfloor} \sum_{t=\max(0,m-2f-2s)}^{\lfloor (m-4f)/3 \rfloor} \sum_{d=\max(0,m-2t-3f-s)}^{\lfloor (m-3t-4f)/2 \rfloor} \frac{s!m!}{2^d (3!)^t (4!)^f d!t!f!(m-2d-3t-4f)!(s-m+d+2t+3f)!}. \tag{3.13}$$

Outer group approximation

We now focus on approximating the number of arrangements in a general outer group.

4 Direct probability approach

Setup

Numbering the cells from 1 to n, we define the random variable X_i to be the number of mines in cell i and we use a slight abuse of notation with $P(X_i) := P(X_i \geq 1)$, in which technically X_i simultaneously acts as the event that at least one mine lies in cell i. We suppose there are r sets of equivalent cells (not including the outer group) and denote a set of equivalent cells as G_j, for $1 \leq j \leq r$, for example a group of four neighbouring cells might be $G_1 = \{11, 12, 13, 14\}$. We also define the random variable representing the total number of mines in the jth group to be Y_j, which is given by $Y_j = \sum_{i \in G_j} X_i$, and we represent the distribution of mines across the groups with the vector $\mathbf{Y} = (Y_j)_{j=1}^r$. We also use the notation $X^{(j)}$ to denote the random variable representing the number of mines in a single cell within group j.

Given a partially solved board, there are a number of valid configurations for the mines in the unclicked cells, corresponding to vectors $\mathbf{y}^{(i)}$ with $P(\mathbf{Y} = \mathbf{y}^{(i)}) > 0$. We use C_i to denote the event that configuration i represents the actual hidden distribution of mines across these groups, which does not take into account the full board situation, so in general $\sum_i P(C_i) < 1$. We then have $P(\mathbf{Y} = \mathbf{y}^{(i)}) = P(C_i) / \sum_j P(C_j)$.

4.1 One mine per cell

We begin with the simple case $x_{\max} = 1$, as in standard minesweeper games, in which case $P(X) \equiv P(X = 1)$. We suppose on the first click of a new game (which we assume to be away from the edges) that we neither hit a mine or an opening, revealing a number α, with $1 \leq \alpha \leq 8$, telling us that the surrounding eight cells contain a total of α mines. We may define this group of cells to be G_1, and we have $P(Y_1 = \alpha) = 1$, with the size of the group being $s_1 = 8$. We have $P(X^{(1)}) = \alpha/8$, giving the safety of all cells in G_1.

We now suppose our first click reveals a number '1' ($\alpha = 1$), which is the most likely case (see the combinatorics approach). The probability of any of the eight surrounding cells containing a mine is $1/8 = 0.125$, which is lower than ρ on any normal game, so the safest move is to choose an adjacent cell. If the first click were to reveal a 2 the probability of a surrounding cell containing a mine is $1/8 = 0.25$ is greater than ρ, so a random cell is safer. We suppose the second click is horizontally adjacent to the previously clicked cell,

and let β represent the number revealed, an example of which is displayed in Figure 2, where we have numbered the cells so that we can take the groups to be $G_1 = \{1, 2, 3\}$, $G_2 = \{4, 5, 6, 7\}$, $G_3 = \{8, 9, 10\}$.

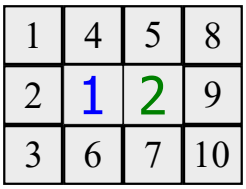
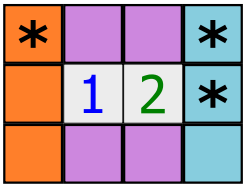
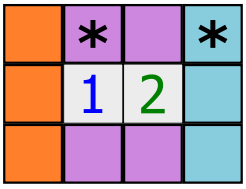


Figure 2: Numbered cells around two clicked cells.

We calculate the probabilities $P(X_i)$ for $1 \leq i \leq 10$ (or equivalently $P(X^{(j)})$ for $j = 1, 2, 3$) by considering all possible configurations of mines in these three groups of cells. There are two configurations for the distribution of mines: $\mathbf{Y} \in \{(1, 0, 2), (0, 1, 1)\}$, which we refer to as configurations 1 and 2 respectively. For C_1 to occur we need exactly



(a) Configuration 1.



(b) Configuration 2.

Figure 3: Groups of equivalent cells represented by colours (numbered from left to right), mines represented by asterisks.

1 mine in a group of 3 cells, none in a group of 4, and 1 in another group of 3.

Approximate solution

To illustrate the calculation we simplify the problem by assuming the grid is very large, so that the mine density, ρ, remains approximately constant. Considering the number of ways the required number of mines can occur in each group and the probability of there being a mine, we get

$$P(C_1) = \left[\binom{3}{1}\rho(1-\rho)^2\right]\left[(1-\rho)^4\right]\left[\binom{3}{2}\rho^2(1-\rho)\right] = 9\rho^3(1-\rho)^7,$$

which corresponds to 3 cells containing mines, 7 cells not containing mines, with the choice of 1 cell in a group of 3, and the choice of 2 cells in another group of 3. Thinking of it this way makes it much easier to just write down probabilities, and we see $P(C_2) = 12\rho^2(1-\rho)^8$. This gives the probabilities for the groups

$$\begin{aligned} P(\mathbf{Y} = (1, 0, 2)) &= \frac{P(C_1)}{P(C_1) + P(C_2)} \\ &= \frac{9\rho^2(1-\rho)^8}{9\rho^2(1-\rho)^8 + 12\rho(1-\rho)^9} \\ &= \frac{3\rho}{3\rho + 4(1-\rho)}, \\ P(\mathbf{Y} = (0, 1, 1)) &= \frac{4(1-\rho)}{3\rho + 4(1-\rho)}. \end{aligned}$$

From this we determine the probability distribution of mines in each of the groups. In this case there is no crossover of the number of mines between configurations, so we have

$$P(Y_1 = 1) = P(Y_2 = 0) = P(Y_3 = 2) = P(\mathbf{Y} = (1, 0, 2))$$
$$P(Y_1 = 0) = P(Y_2 = 1) = P(Y_3 = 1) = P(\mathbf{Y} = (0, 1, 1)).$$

Finally, we find $P(X_i)$ for each cell i by considering the expected value of $\mathbf{Y}$ and dividing by the size of the group. Taking $\rho = 0.207$ (expert board with two cells clicked), we calculate the probabilities for the situation in Figure 2:

$$P(X^{(1)}) = \frac{1}{3}\left(\frac{3\rho}{3\rho + 4(1-\rho)}\right) \approx 0.055,$$
$$P(X^{(2)}) = \frac{1}{4}\left(\frac{4(1-\rho)}{3\rho + 4(1-\rho)}\right) \approx 0.209,$$
$$P(X^{(3)}) = \frac{1}{3}\left(\frac{6\rho + 4(1-\rho)}{3\rho + 4(1-\rho)}\right) \approx 0.388.$$

0.055	0.209	0.209	0.388
0.055	**1**	**2**	0.388
0.055	0.209	0.209	0.388

So we conclude that the best next move would be to click on one of the side cells on the left where there is a low chance of hitting a mine.

Refinements

To obtain an exact answer, we drop the assumption that ρ is constant, so that choosing one cell to be safe makes the others slightly more likely to contain a mine. Going back to the first configuration as in Figure 3a, we calculate the probability to be

$$P(C_1) = \binom{3}{1}\binom{3}{2}\prod_{j=1}^{3}\left(\frac{K-j+1}{N'-j+1}\right)\prod_{j=1}^{7}\left(1 - \frac{k-3}{n-j-2}\right)$$
$$= \binom{3}{1}\binom{3}{2}\frac{K_{(3)}(N'-K)_{(7)}}{N_{(10)}},$$

where $x_{(a)} = x!/(x-a)!$ is the falling factorial, and N' is the remaining number of clickable cells (in this case $N' = N - 2$). For the start of an expert game this new equation gives the adjusted values a shown below.

0.044	0.217	0.217	0.378
0.044	**1**	**2**	0.378
0.044	0.217	0.217	0.378

In general we have a set of groups $\{G_i : 1 \leq i \leq r\}$ with size $|G_i| = s_i$, which span $S = \sum_{i=1}^{r} s_i$ cells and have various mine configurations C_j with a total of M_j mines. Suppose the number of mines in group G_i, configuration C_j, is m_{ij}, where $M_j = \sum_{i=1}^{r} m_{ij}$, then the relative probability of a configuration occurring is given by

$$\xi(C_j) = \frac{K_{(M_j)}(N'-K)_{(S-M_j)}}{N'_{(S)}}\prod_{i=1}^{r}\binom{m_{ij}}{s_i}. \tag{4.1}$$

Since this is normalised by dividing by the sum of probabilities for all configurations to get the distribution of $\mathbf{Y}$, it is valid to drop the constant factor $N'_{(S)}$ to give the relative probability

$$\xi(C_j) = K_{(M_j)}(N' - K)_{(S-M_j)} \prod_{i=1}^{r} \binom{m_{ij}}{s_i}, \tag{4.2}$$

as seen in Section 3. From this we compute the individual probabilities using the expected number of mines in a configuration, giving

$$P(X^{(i)}) = \frac{1}{s_i} \sum_j x_j P(Y_i = x_j). \tag{4.3}$$

4.2 Multiple mines per cell

In the case $x_{\max} \geq m_1$ for a group of size s_1 which contains m_1 mines, we think of the mines being allocated randomly to the cells, which gives $X^{(1)} \sim \text{Bin}(m_1, 1/s_1)$ and $P(X^{(1)}) = 1 - ((s_1 - 1)/s_1)^{m_1}$. This equation provides a lower bound for the probability for all $x_{\max}$, and the method above for $x_{\max} = 1$ provides an upper bound. The lower bound can be used as an approximation under certain circumstances. To calculate the probability of a cell containing a mine for $x_{\max} \leq m_1$ ...

We note that the only change we would see for $x_{\max} > 1$ is for the groups which can contain more than one mine. We return to the case considered in Section 4.1 where only G_3 can contain more than one mine. We would now obtain a lower value for $P(X^{(3)})$ – there is a chance both mines are in one cell, meaning the chance of each cell being safe is a little higher.

5 Application to whole-board calculations

So far we have the following process for calculating the probabilities of a general partially-solved board:

1. Find the equivalence groups, G_i, of cells next to a revealed number (it would be just as valid to treat each cell as its own 'group' – grouping into equivalent cells is a useful optimisation).

2. Find all possible configurations, C_j, of mines in the equivalence groups.

3. For each configuration calculate the relative probability, $\xi(C_j)$, of it being the actual configuration, given by (3.10)? (which also requires the calculation of the number of ways to arrange mines within equivalence groups, w_{ij}, given by (3.12)?) or (4.2)?.

4. Divide the relative probabilities by the sum of all relative probabilities to obtain the actual probabilities of each configuration occurring.

5. Given a configuration with a fixed number mines in each group, for each group calculate the probability of a single cell within the group containing at least one mine (using ?). Sum up these probabilities for each configuration, weighting by the probability of the configurations occurring. This results in the probability of each cell containing at least one mine (for the cells which neighbour a revealed number).

To calculate the probability for cells in the 'outer' equivalence group it could be included as one of the groups, and therefore included in the configurations. The number of mines in the outer group would be determined by subtracting the number contained in the rest of the groups for a given configuration from the total number on the board, K (so the total for each configuration would be fixed at K). However, since calculations for large groups can take longer than desired, this can be approximated by calculating the expected number of mines across all of the configurations, and subtract this number (rounded to the nearest integer) from K to give an approximation for the number of mines in the outer group. With this approach the probabilities for each individual cell only need to be calculated once (this is discussed in greater detail in ?).

In this section we discuss each of these steps in greater detail with regards to a formal algorithm, how it can be implemented using pseudo-code, and what optimisations and/or approximations can be used.

5.1 Finding equivalence groups

5.2 Finding configurations

Splitting up into islands

Consider the following 'simplest case', depicted in Figure 4. There are two distinct areas

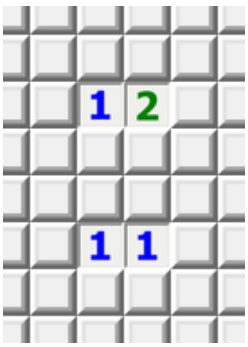


Figure 4

which have no direct influence on each other – the only influence is on the density of mines. Therefore their configurations can be found independently.

The two islands have the following sets of configurations: $I_1 = \{(1,0,2),(0,1,1)\}$ and $I_2 = \{(1,0,1),(0,1,0)\}$. Therefore we have a total of $2+2=4$ configurations of length 3, whereas if we hadn't split up into islands we would have $2\times 2=4$ configurations of length 6. Since the number of possibilities for each island is small in this case, we coincidentally end up with the same number of configurations in either case, but note the difference of adding and multiplying, meaning that in general splitting up into islands will result in many less configurations needing to be found (or stored in memory). For example a board with three islands with 2, 10 and 8 configurations respectively, we reduce 160 configurations to just 20 (and of shorter length).

5.3 Calculating relative probabilities for configurations

This is the step which contains the most mathematics, which has been discussed extensively in Sections 3? and 4?. Here we extend the discussion from these previous

sections by summarising the general method, in particular when dealing with 'islands' as introduced in Section 5.2.

Firstly we recall the formula we previously found for the relative probabilities of the configurations:

$$\xi(C_j) = \frac{\tilde{w}_j}{(K - M_j)!} \prod_{i=1}^{r} \frac{w_{ij}}{m_{ij}!}, \tag{5.1}$$

where for $x_{\max} = 3$ we have

$$w = \sum_{t=\max(0,m-2s)}^{\lfloor m/3 \rfloor} \sum_{d=\max(0,m-2t-s)}^{\lfloor (m-3t)/2 \rfloor} \frac{s!m!}{2^d (3!)^t d!t!(m-2d-3t)!(s-m+d+2t)!}. \tag{5.2}$$

This formula applies when groups aren't split into islands, so for the example Figure 4 we would have the configurations

$$\{C_1, C_2, C_3, C_4\} = \{(1,0,2,\ 1,0,1),\ (1,1,1,\ 1,0,1),\ (1,0,2,\ 0,1,0),\ (1,1,1,\ 0,1,0)\},$$

with K and N assumed to be known.

The form of (5.1) is a product term concerned with the groups within the configuration multiplied by a term concerned with the outer group, or in general the rest of the groups which are not included in the configuration. We may split the relative probabilities formula by defining

$$\zeta(C_k^l) = \prod_{i=1}^{r_l} \frac{w_{ikl}}{m_{ikl}!}, \qquad \eta_j = \frac{\tilde{w}_j}{(K - M_j)!} \tag{5.3}$$

where C_k^l is the kth configuration of the lth island, and η is the factor considering the outer group which depends on the total number of mines in the configurations on each island. In the case where we don't split into islands we have $C_j = C_j^1$ for all j, so we may rewrite the relative probability for a configuration as

$$\xi(C_j) = \eta_j \zeta(C_j^1). \tag{5.4}$$

We are now free to split the configurations into the islands

$$I_1 = \{C_1^1, C_2^1\} = \{(1,0,2),\ (0,1,1)\}, \qquad I_2 = \{C_1^2, C_2^2\} = \{(1,1,1),\ (0,1,0)\},$$

where we can take C_1 to correspond to C_1^1 and C_1^2, C_2 to correspond to C_2^1 and C_1^2 etc., as above for C_3 and C_4 also. This allows us to write the relative probability for the full configurations C_j as

$$\xi(C_j) = \eta_j \prod_l \zeta(C_{k_{jl}}^l), \tag{5.5}$$

where the index of the chosen configuration within an island is determined by which configuration we're looking at (the value of j) and of course which island (the value of l). It's easy enough to come up with a systematic mapping from the island configurations to the full configurations, which would fully determine the k_{jl}, however this is unimportant for now and will naturally arise when translating the method into code.

5.4 Mapping configuration probabilities to cell probabilities

So far we have removed the necessity of calculating the full set of configurations as they are split into possibilities within islands. However, eventually this cannot be avoided since to calculate the probabilities for configurations in an island we need to know how many mines are placed across the board, i.e. in configurations of other islands (in fact we may have created configurations on different islands which are incompatible because they use more mines than the whole board contains).

Configuration probabilities to group probabilities

We must now loop through the islands, reconstructing the full configurations, although the hope is that we will have saved a lot of computing time by not actually creating and storing arrays for the full-length configurations. Continuing with the above example we now illustrate how the ζ function is used to find the relative probabilities for the full configurations.

```
INITIALISE configProbs array to contain probabilities of each configuration
FOR EACH config in the first island:
    CALCULATE zeta of the config in the first island
    FOR EACH config in the second island:
        CALCULATE zeta of the config in the second island
        CALCULATE eta
        CALCULATE xi by multiplying zeta values and eta
        STORE xi in configProbs array
```

Note $\zeta(C_k^2)$ is being calculated more than once for each k, which could easily be changed (by calculating the ζ function for each configuration in each island before starting the embedded loops used to access full configurations). We now find the actual probability for each configuration by dividing by the sum of the relative probabilities as in (?).

```
CALCULATE sum of all configProbs
FOR EACH prob in configProbs:
    DIVIDE prob by the sum
```

Now that we know the actual probability of each configuration occurring, we could revert back to considering islands separately. In fact we also have the choice of operating with `configProbs` and accessing the mines assigned to the island configurations using manipulation of the index, and this may produce better performance if the number of configurations is small. However it seems clearer to consider groups within their islands, since once we know the configuration probabilities islands have no effect on each other.

```
INITIALISE islandProbs arrays to contain config probabilities by island
FOR EACH prob in configProbs:
    FOR EACH islandConfig used in the full config with probability prob:
        INCREMENT corresponding element of islandProbs by prob
```

This leaves us with isolated islands which will usually have a fairly small number of configurations (almost certainly less than 100 in a single island), which makes the rest of the code logic fairly quick to run. We now need to consider groups individually and calculate the probability of a cell within a group containing at least one mine. We do this island by island.

```
FOR EACH group in an island:
    INITIALISE groupProb as 0
    FOR EACH config in the same island:
        CALCULATE cellUnsafeProb for the cells in group
        MULTIPLY cellUnsafeProb by probability of config
        INCREMENT groupProb by cellUnsafeProb
```

The line to pay attention to in the above code is the one calculating `cellUnsafeProb`, which involves the mathematics discussed briefly above and can be the most time consuming part of the above code. We now go into more detail about how this function works...

Group probabilities to cell probabilities

As mentioned briefly in Section ?,

5.5 Further optimisations

Deducing certainties

Dealing with large numbers of combinations

For a large board we find the number of combinations gets very large – that is $\xi(C_j)$ can become a very large number. For example on an expert board which has barely been started we have $\xi \sim \binom{480}{99} \sim 10^{924}$ for $x_{\max} = 1$ (and higher for higher $x_{\max}$). Numbers of this size can cause problems when trying to use them in programming languages[2], as well as being slow since we are dealing with integers and therefore exact precision is maintained, which is entirely unnecessary.

One trick to tackle this problem is to convert using the exp and log functions, changing multiplication and division to addition and subtraction respectively. For example, instead of calculating ξ, we calculate $\log \xi$ as follows:

$$\bar{\xi}(C_j) := \log \xi(C_j) = \log \eta_j + \sum_l \log \zeta(C^l_{k_{jl}}), \tag{5.6}$$

and as such instead of calculating and storing ζ for each configuration we would instead use

$$\log \zeta(C^l_k) = \sum_{i=1}^{r} \log(w_{ikl}) - \log(m_{ikl}). \tag{5.7}$$

This should perform better in terms of speed, if for no other reason then just because exact precision is no longer maintained when using floating point decimals (which could have been achieved simply by defining the functions to use floating point decimals). However to find the probability of the configurations we would then need to sum up the values for ξ, meaning we would use

$$P(C_j) = \frac{\exp(\bar{\xi}(C_j))}{\sum_i \exp(\bar{\xi}(C_i))}, \tag{5.8}$$

[2]Using one of the more 'friendly' programming languages as an example, Python 3.6 has no limit on the size of an integer so it can cope with these calculations, however to maximise speed floats should be used and their standard maximum limit is $\sim 10^{308}$ which is insufficient.

and the large numbers arise once again.

Another approach is to remove common factors from the relative probability ξ, since it is only needed to be relative to the other configurations. There is an obvious way to remove factors from the formula for w – for a given board situation the sizes of the groups are fixed, so $s!$ can be removed from equation (5.2)? without changed the result (again, this is because we are dealing with relative probabilities). There is a reason to be cautious about this – as it currently stands ξ is a function which is guaranteed to return an integer, with some direct correspondence to the number of configurations, and this may be advantageous if we're looking to produce very accurate results or if it turns out to be quicker for a programming language to be dealing with integers. We could get around this by diving w by a falling factorial of s (constant across all configurations) which would still leave a factor in the numerator to cancel out the $(s-m+d+2t)!$ term in the denominator.

Since there is no need for us to keep ξ as a function returning an integer it is recommended to redefine w without the $s!$ factor, and it remains unclear whether use of logs and exponentials can offer (further) speed enhancements.

6 Conclusion and related problems

A good opening strategy to find an opening is as follows:

1. Click randomly in the middle of the grid (20% chance of losing).
2. If a number 1 is revealed, click an adjacent cell, say the one to the right (13% chance of losing).
3. If we now have a 1-1 configuration, click one of the six cells adjacent to only one of the revealed '1's, say the one directly to the right (12% chance of losing). If we have a 1-2 configuration, click one of the three cells only next to the 1, say the one directly to the left (5% chance of losing).
4. If we now have a 1-1-1 configuration, click one of the 6 cells on the ends, next to only one of the '1's (10% chance of losing). If we have a 1-1-2 configuration, click one of the two cells shared by only the two '1's (6% chance of losing), or maybe click one of the 3 cells on the far left for a higher chance of getting an opening?? (13% chance of losing).

Interesting mathematical questions related to minesweeper include:

- What is the probability of solving the grid if we play optimally?
- What is the probability of being able to solve the grid without guessing, if we are given that the first click reveals an opening?
- What is the expected number of mines that would be hit if we had infinite lives, if again we were to play optimally?

Perhaps a reasonable simplification would be to treat the grid as infinite with a mine density of ρ and attempt to calculate the above problems for an area of cells.

Appendices

A Python code

First we define the functions ζ and w:

```
from math import factorial as fac
def zeta(cfg, groups, xmax):
    result = 1
    for i, g in enumerate(groups):
        result *= w(m, s, xmax) / fac(cfg[i])
    return result
def w(m, s, xmax):
    assert xmax in [1, 2, 3]
    if xmax == 1:
        return fac(s) / fac(s - m)
    max_trip_mines = m // 3 if xmax == 3 else 0
    initial = fac(s) * fac(m)
    result = 0
    for t in range(max(0, m - 2*s), max_trip_mines + 1):
        for d in range(max(0, m - 2*t - s), (m - 3*t) // 2 + 1):
            result += initial / (
                2**d * 6**t * fac(d) * fac(t) *
                fac(m - 2*d - 3*t) * fac(s - m + d + 2*t))
    return result
```

Next we find the number of full configurations and calculate and store ζ for each island configuration (to save calculating it many times in embedded loops).

```
## Assume we have a list, all_islands, of island dicts such as
## {'configs': [(0,1,...), ...],
##  'nConfigs': 2,
##  'config_probs': [0,0,...], #length nConfigs
##  'groups': [...]
##  }
# Find number of configs
nConfigs = 1    #assume not zero
for i, island in enumerate(all_islands):
    nConfigs *= island['nConfigs']
    # Find results of zeta function for each config
    island['zetas'] = []
    for cfg in island['configs']:
        ## Somewhere xmax is defined
        z = zeta(cfg, island['groups'], xmax)
        island['zetas'].append(z)
```

Next loop through the island configurations using an array storing positions (think of a digital clock), and store the full configuration probabilities in an array.

```
config_probs = [0] * nConfigs
positions = [0] * len(all_islands)
j = 0
```

```
while positions[-1] < all_islands[-1]['nConfigs'] - 1:
    prob = 1
    mines = 0
    for l, island in enumerate(all_islands):
        z = island['zetas'][positions[l]]
        prob *= z
        cfg = island['configs'][positions[l]]
        mines += sum(cfg)
    ## Defined somewhere is outer_cells, the size of the outer group
    eta = w(K - mines, outer_cells, xmax) / fac(K - mines)
    prob *= eta
    config_probs[j] = prob
    j += 1
    for i, pos in enumerate(positions):
        if pos == all_islands[i]['nConfigs'] - 1:
            positions[i] = 0                        //reset to zero
        else:
            positions[i] += 1                       //increment...
            break                                   //...and quit loop
total = sum(config_probs)
for j in range(nConfigs):
    config_probs[j] /= total
```

As before we then split back up into island configuration probabilities, again using the 'digital clock' counter.

```
# Reset positions counters
for i in range(len(positions)):
    positions[i] = 0
j = 0
while positions[-1] < all_islands[-1]['nConfigs'] - 1:
    prob = config_probs[j]
    for l, island in enumerate(all_islands):
        pos = positions[l]
        island['config_probs'][pos] += prob
    j += 1
    for i, pos in enumerate(positions):
        if pos == all_islands[i]['nConfigs'] - 1:
            positions[i] = 0
        else:
            positions[i] += 1
            break
```

Finally we calculate the probability of there being a mine within the individual cells which belong to the groups. We loop over each of the islands contained in `all_islands` (the logic for a single island is shown here).

```
def get_unsafe_prob(m, s, xmax):
    assert xmax in [1, 2, 3]
    if m > s*xmax:
        raise ValueError("Invalid configuration")
    if xmax == 1:
        return m / s
```

```
    elif xmax >= m:
        return 1 - (1 - 1/s)**m
    elif m > xmax*(s - 1):
        return 1
    else:
        return 1 - w(m, s - 1, xmax) / w(m, s, xmax)
nGroups = len(island['configs'][0])    #find number of groups
for i in range(nGroups):               #loop through the groups
    group = island['groups'][i]
    for k, cfg in enumerate(island['configs']):
        ## Each group is a dictionary such as:
        ## {'coords': [(1,2), (2,2)],
        ##  'size': 2, #length of list of coords
        ##  'probs': [0,0,...], #length size*xmax+1 for nr of mines
        ##  'cell_unsafe_prob': 0,
        ##  ...}
        mines = cfg[i]
        cfg_prob = island['config_probs'][k]
        group['probs'][mines] += cfg_prob
        unsafe_prob = get_unsafe_prob(mines, group['size'], xmax)
        group['cell_unsafe_prob'] += unsafe_prob * cfg_prob
```

The coordinates stored in the group dictionaries are unique to their group, so the `cell_unsafe_prob` key can then be used to determine the probability of each of the cells in the grid which are next to a revealed number.

References

[1] R. Kaye, Minesweeper is NP-complete, *The Mathematical Intelligencer* 22(2) (2000) 9-15.